Lecture 24 :- Flow Control

Everywhere except switch curly braces is optional

1. Selection Statement
   1. if-else
   2. switch()

2. Iterative Statement
   1. while()
   2. do-while()
   3. for()
   4. for-each loop (1.5V)

3. Transfer statement.
   1) break
   2) continue
   3) return
   4) try-catch-finally
   5) assert (1.4V)

CLASSTIME Pg. No.
Date / /

Flow Control describe the order in which the statements will be executed at Runtime.

(1) Selection Statement :- Among several options only one option will be selected and executed.

(2) Iterative Statement :- A group of statements will be executed iteratively.

(3) Transfer Statements :- Statement transfer the control.

(1) Selection Statement :-
if - else :-

Syntax :-

```
if (b)
{
   Action if b is true
}
else
{
   Action if b is false.
}
```

Should be boolean type (argument)

The argument to the if statement should be boolean type by mistake if you are

trying to provide any other type then we will get CTE.

```
int x=0;
if(x)
{
  sopln("Hello");
}
else
{
  sop("Hi");
}
```

```
int x=10;
if(x==20)
{
  sopln("Hello");
}
else
{
  sop("Hi");
}
```

CE: incompatible types
found: int
req: boolean.

```
int x=10;
if(x==20)
{
  sopln("Hello");
}
else
{
  sopln("Hi");
}
```

O/P: Hi

```
boolean b=true;
if(b=false)
{
  sopln("Hello");
}
else
{
  sopln("Hi");
}
```

O/P: Hi

CLASSTIME | Pg. No.
Date / /

```
boolean b = false;
if (b == false)
{
    sopln ("Hello");
}
else
{
    sopln ("Hi");
}
```

O/p:- Hello.

Conclusion 2: else part and curlibraces are optional. without curlibraces only one statement is allowed under if which should not be declarative statement.

```
if (true)
    sopln ("Hello");
```

```
if (true);
```

```
if (true)
{
    int x = 10;
}
```

CE:

```
if (true)
{
    int x = 10;
}
```

Note: Semicolon is a valid Java statement which is also has empty statement.

Note: There is no dangling else problem in Java. Every else is matched/mapped to the nearest if statement.

```
if (true)
    if (true)
        sopln ("Hello");
    else
        sopln ("hi");
```

nearest if is mapped else.

(2) Switch(x) statement :- If several options are available then it is not recommended to use nested if-else. because it reduces readability. to handle this requirement we should go for switch statement.

Syntax:-

```
switch (x)
{
    case 1:
        Action - 1;
        break;
    case 2:
        Action-2;
        break;
    :
    case n:
        Action-n;
        break;
    default:
        default Action.
}
```

CLASSTIME Pg. No.
Date / /

Loophole 1 :- The allowed argument types in for the Switch statement are byte, short, char, int until 1.4 v. But from 1.5 v onwards corresponding wrapper classes and enum type also allowed. from 1.7 v onwards String type also allowed.

1.4V	1.5V	1.7V
byte short char int	Byte Short Character Integer + Enum	String

1) Boolean condition — X → only true/false so only 2 is allowed.

2) long — very long value not reqrd.

float — infinite no. of cases/value.
double —

Loophole 2 :-

```
Switch (x)
{
}
```

mandatory.

CLASSTIME Pg. No.
Date / /

1) ★ Curlibraces are mandatory. except Switch everywhere curlibraces { } are optional.

2) ★ Both Case and default are optional. i.e. an empty Switch statement is a valid Java Syntax.

```
int x=10;
Switch(x)
{
}
```

✓

3) Inside Switch every statement should be under some Case or default. i.e. independent statements are not allowed inside Switch otherwise we will get CTE.

```
int x=10;
Switch(x)
{
  System.out.println("Hello");
}
```

CE: Case, default, or } expected.

(4) Every Case labeled should be Compile time Constant (i.e. Constant expression).

Eg: `int x=10;`
`int y=20;`

```
switch (x)
{
   case 10:
      sopln(10);
      break;
   case y:
      sopln(20);
      break;
}
```

CE: Constant expression required.

* If we declare Y as final then we won't get any CTE.

* Both switch argument and case label can be expressions. But case label should be Constant expression.

Ex:
```
int x=10;
switch (x+1)
{
   case 10:
      sop(10);
      break;
   case 10+20+30:
      sopln(60);
}
```

expression → int + int

exp / but const

CLASSTIME Pg. No.
Date / /

Every case labelled should be in the range of switch argument type. Otherwise we will get CTE.

ex: byte b = 10;
switch (b)
{
case 10:
sop(10);
break;
case 100:
sop(100);
break;
case 1000:
sop(1000);
}

byte: -127 to 127.

but got 1000.

CE: possible loss of precision
found: int
req: byte

---

* byte b = 10;
switch (b+1) → int.
{
case 10:
sop(10);
break;
case 100:
sop(100);
break;
case 1000:
sop(1000);
}

```
int x=10;
switch(x)
{
  case 97:
      sopln(97);
      break;
  case 98:
      sop(90);
      break;
  case 99:
      sopln(99);
      break;
  case 'a':
      sopln('a');
}
```

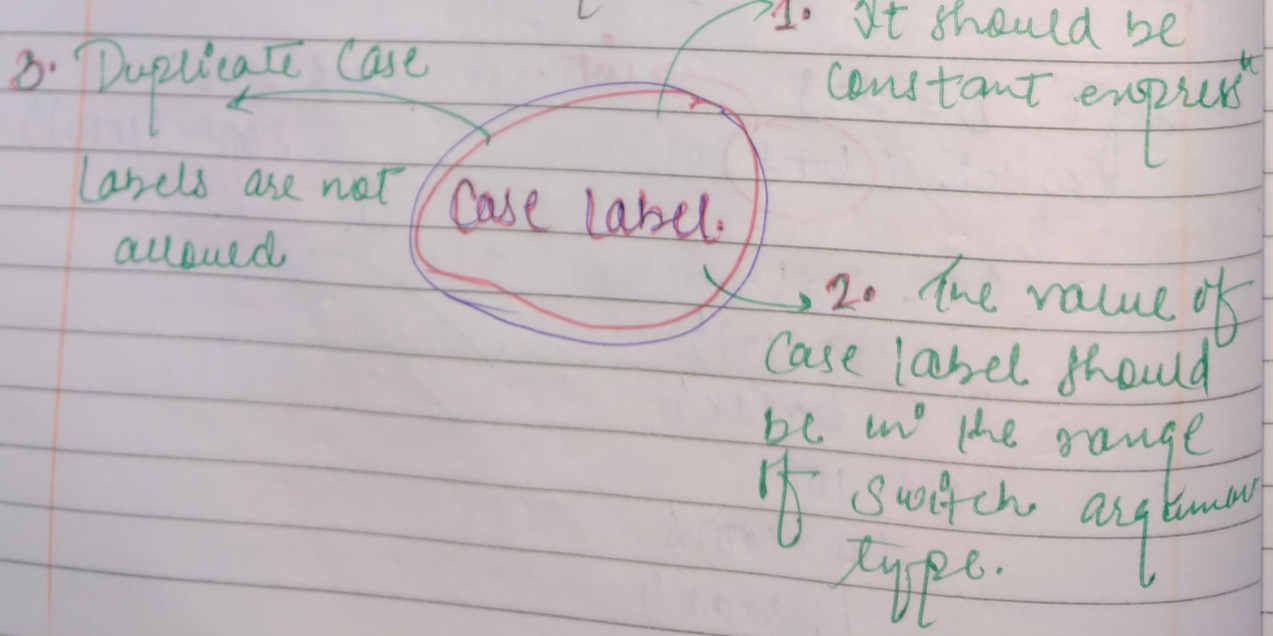
Ambiguity

97 ← value.

3.

duplicate case labels are not allowed.
↳ otherwise we will get CTE.
CE: Duplicate case label

* Case label Summary :-



CLASSTIME Pg. No.
Date / /

Fall through inside Switch :-

With in the Switch if any case is matched from that case onwards all statements will be executed until break or end of the Switch. this is called fall through inside a Switch.

The main advantage of fall throug inside a Switch is we can define common action for multiple cases. (Code reusability).

```
Switch (x)
{
  Case 1:
  Case 2:
  Case 3:
    Sopln("Q-4");
    break;

  Case 4:
  Case 5:
  Case 6:
    Sopln("Q-1");
    break;
  :
  :
}
```

```
Switch (x)
{
    case 0:
        sopln(0);
    case 1:
        sopln(1);
        break;
    case 2:
        sopln(2);
        break;
    default:
        sopln("def");
}
```

O/p:

Case x=0	Case x=1
0	1
1	
Case x=2	Case x=3
2	
def.	def

default Case:- 1) With the switch we can take default case atmost once.

2) default case will be executed if there is no case matched.

3) With the switch we can write default case anywhere but it is recommended to write as last case.

```
Switch (x)
{
    default:
        sopln("def");
    case 0:
        sopln(0);
        break;
}
```

O/p:

Case x=0	Case x=1
0	1
	2

```
case 1:
    System.out.println(1);
case 2:
    System.out.println(2);
}
```

case x=2	case x=3
2	def
	0

Lecture 27 :- Iterative Statements

1) while()
2) do-while().
3) for()
4) for-each(loop.

(1) **while()** :- If you don't know no. of iterations in advance then we should go for while loop.

```
ex:- while(rs.next())
     {
     :
     }
```

```
while(e.hasMoreElements())
    {
    ==
    }
```

```
while(itr.hasNext())
    {
    ==
    }
```

Syntax:-

```
while (b)
{
    action;
}
```

loop body

b should be boolean type.

The argument should be boolean type if we are trying to provide any other type then we will get CTE.

```
while (1)
{
    sopln("Hello");
}
```

1 → treated as value. not boolean in Java.

O/p:-

CE: incompatible types
found: int
required: boolean.

`while (true) sopln("Hello");` ✓	`while (true);` ✓
`while (true) int x=10;` declaration ✗	`while (true) { int x=10; }` ✓

* Curlibraces are optional and without Curlibraces we can take only one statement after while which

should not be declarative statement.

①
```
while (true)
{
    sopln("Hello");
}
sopln("Hi");
```
CE: Unreachable statement.

②
```
while (false)
{
    sopln("Hello");
}
sopln("Hi");
```
CE: Unreachable statement '{'.

③
```
int a=10, b=20;
while (a<b)
{
    sopln("Hello");
}
sopln("Hi");
```
o/p: Hello
Hello
:
∞

④
```
int a=10, b=20;
while (a>b)
{
    sopln("Hello");
}
sopln("Hi");
```
o/p: Hi

↳ Isme int variable ki value change ho sakti hai kyo ki ye normal variable hai. and isme kyo ki compiler reachability check karega tab jab value ko consider karega but uyaha par runtime pe value change hone ki chance hai and compiler value ko ignore karega and JVM only execute karta rahega kyo ki uska wahi kaam hai.

(5) final int a=10, b=20;
while (a<b) — true always
{
sopln("Hello");
}
sopln("Hi");

CE: Unreachable Statement

(6) final int a=10, b=20;
while (a>b) — false always
{
sopln("Hello");
}
sopln("Hi");

CE: Unreachable statement

Note: (1) Every final variable will be replaced by the value at CT only. So why we should wait for R. till R runtime.

(1) eg: final int a=10;
int b=20;
sopln(a); → sop(10);
sopln(b); → sop(b);
(After Compilation)

Note (2): If every argument is a final variable (compile time constant) then that operation should be performed at compile time only.

ex: final int a=10, b=20;
int c=30;
sop(a+b); → sopln(30);
sopln(a+c); → sop(10+c);

sopln (a<b);
sopln (a<c);

Computation ⇒

sopln (true);
sopln (10<20);

* do-while:- If we want to execute loop body atleast Once then we should go for do-while.

Syntax:-

```
do
{ --optional
  body
} while (b);
```

→ mandatory in Java.
↳ should be boolean type.

Curli braces are optional and without Curli braces we can take only one statement b/w do and while which should not be declarative statement.

① do
  sop ("Hello");
  while (true);

② do;
  while (true);

③ do
  int x=10; ✗ declare
  while (true);

④ do
  {
  int x = 10;
  } while (true);

⑤ do
  while (true);
  ✗

```
do while(true)
sopln("Hello");
while(false);
```

do-while loop

```
do
  while(true)
    sopln("Hello");
while(false);
```

O/p: Hello
Hello
Hello
!
∞

Single try Catch block →

```
try
{
  // Lakh Lines code
}
catch
{
  // Lakh Lines code
}
finally
{
  // Lakh Lines code
}
```

Valid do-while Syntax above don't keep any doubt.

Unreachability :-

① do-while loop

```
do
{
  sop("Hello");
}
while(true);
sopln("Hi");
```

CE: Unreachable Statement

② execute once (do)

```
do
{
  sopln("Hello");
}
while(false);
sopln("Hi");
```

O/p:- Hello
Hi — only one

③ int a=10, b=20;
do
{
loop → sopln("Hello");
} while (a<b); → true
sopln("Hi"); → not execute.

o/p:
Hello
Hello
:
∞

④ int a=10, b=20;
do
{
① sopln("Hello"); → exect ① → false
} while (a>b);
① sopln("Hi"); → exect ①

o/p: Hello ✓
Hi ✓

⑤ final int a=10, b=20;
do
loop → {
sopln("Hello"); → true
} while (a<b);
sopln("Hi");

→ CE: Unreachable statement

⑥ final int a=10, b=20;
do
{
sopln("Hello");
} while (a>b); → false
sopln("Hi");

o/p: Hello
Hi.

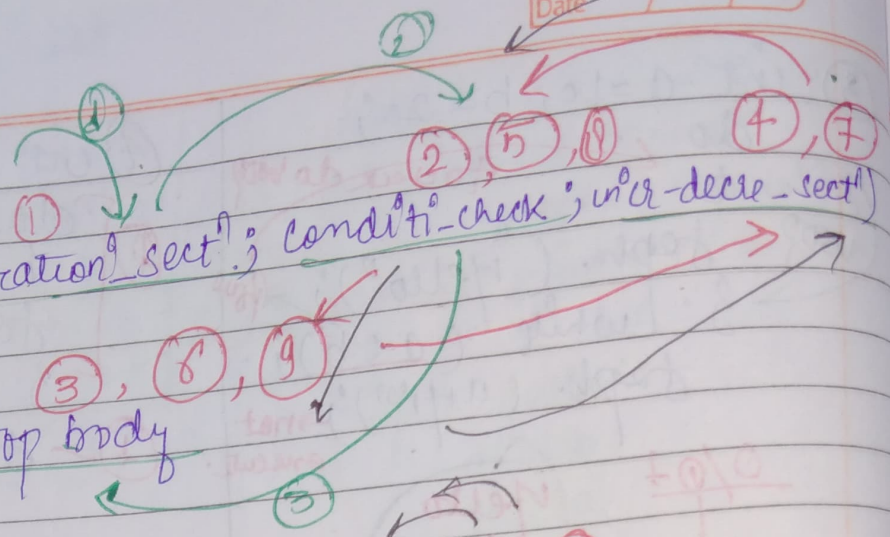
Lecture-28 : for & for-each loop.

(i) for loop: For loop is the most commonly used loop in java.

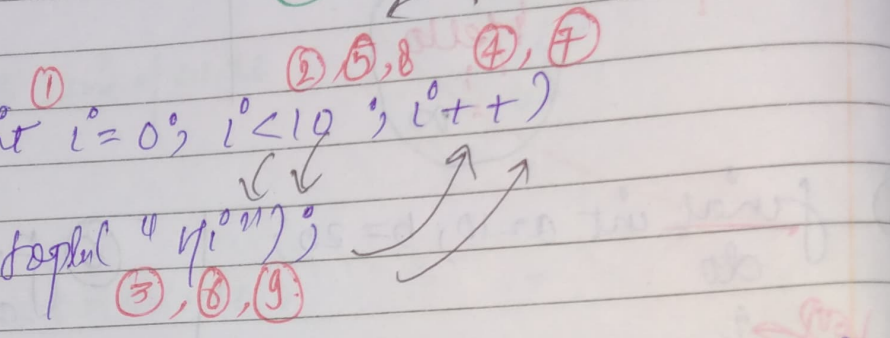
(ii) If we know no. of iterations in advance then for is the best choice.

CLASSTIME Pg. No.
Date / /

Syntax :-



eg :-



Curlibraces :- { } :- Curlibraces are optional and without curlibraces we can take only one statement under for loop, which should not be declarative statement.

eg :- (1) for (int i=0; true; i++)
sopln ("hello");

(2) for (int i=0; i<10; i++);

(3) for (int i=0; i<10; i++)
int x=10; ✗

(1) Initialization section :-

(2) This part will be executed only once in loop lifecycle.

(3) Here we can declare and initialize local variables of for loop.

(4) Here we can declare any no of variables but should be of the same type., By mistake if we are trying to declare different datatype variables then we will get CTE.

Eg:- int i=0, j=0; ✓

int i=0, String s="durga"; ✗

int i=0, int j=0; ✗

(4) In the initialization section we can take any valid Java statement including sopln.

Eg:- int i=0;

for(sop("Hello boss U are sleeping"); i<3; i++)

{

sopln("NO Boss U only sleeping");

}

O/p:-

Hello Boss U R sleeping

NO Boss U only sleeping

NO Boss U only sleeping

NO Boss U only sleeping.

CLASSTIME Pg. No.
Date / /

Conditional check :- Here we can take any valid Java expression but should be of the type boolean.

This part is optional and if we are not taking anything then compiler will always place true.

eg :- for (int i=0; true ; i++) (by default)
{
sop ("Hi");
}

* Increment/decrement Section :-

In the increment & decrement section we can take any valid Java statement including sop's.

eg :- int i=0;
for (sop ("Hello"); i<3; sop ("Hi")) ;
{
i++;
}

Op :-
Hello
Hi
Hi
Hi

All three parts of for loop are independent of each other and optional.

```
for ( ; ; )
{
    sopln ("Hello");
}
```

```
for ( ; ; );
```

Infinite loops

Unreachability :-

```
for( int i=0; true; i++)
{
    sopln ("Hello");
}
sopln ("Hi");
```

Unreachable Statement.

```
for( int i=0; false; i++)
{
    sopln ("Hello");
}
sopln ("Hi");
```

O/P: Unreachable Statement '{'.

```
for( int i=0; ; i++)
{
    sopln ("Hello");
}
sopln ("Hi");
```

O/P: Unreachable Statement.

```
int a=10, b=20;
for( int i=0; a<b; i++)
{
    sopln ("Hello");
}
sopln ("Hi");
```

O/P :-
Hello
Hello
;
;

CLASSTIME Pg. No.
Date / /

```
int a=10, b=20;
for(int i=0; a>b; i++)
{
    sopln("Hello");
}
sopln("Hi");
```

Op: Hi

```
final int a=10, b=20;
for(int i=0; a<b; i++)
{
    sopln("Hello");
}
sopln("Hi");
```

CE: Unreachable statement.

```
final int a=10, b=20;
for(int i=0; a>b; i++)
{
    sopln("Hello");
}
sopln("Hi");
```

CE: Unreachable statement
'{'

For-each Loop :- (Enhanced for loop) :-

(1) 1.5 version ° introduced in 1.9.

2) Specially designed for Arrays & Collection°. to retrive elements of arrays & Collection

Case 1 :- To print elements of 1-D array :-

```
int[] x = {10,20,30,40};
```

Normal for loop	Enhanced for loop
for(int i=0; i<x.length; i++) { sopln(x[i]); }	for(int x1 : x) { sopln(x1); }

for each int value of x1 in x
Har ek int value x1 in x.

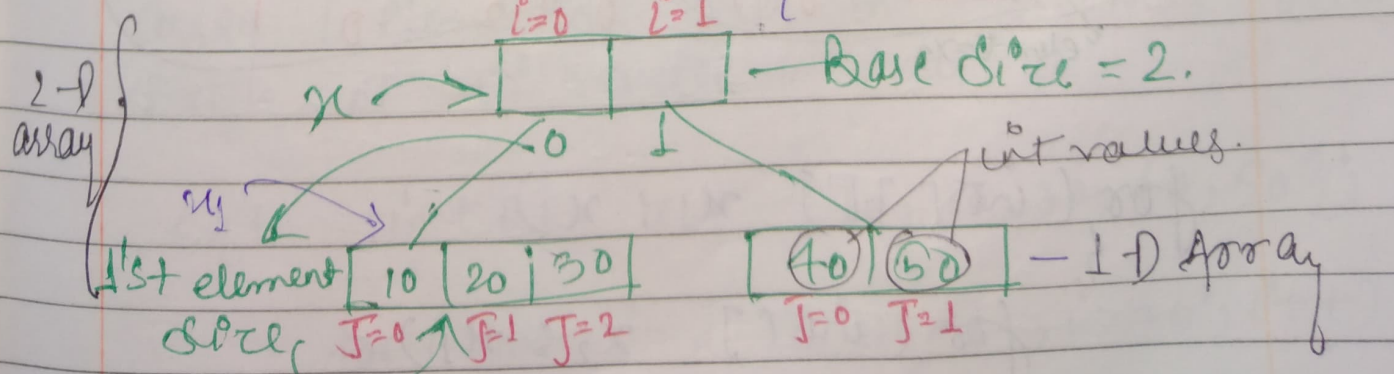
x → [10 | 20 | 30 | 40]

x1 (element)

ex :- To print elements of 1-D Array.

Ex-2 :- To print elements of 2-D Array :-

```
int[][] x = {{10,20,30},{40,50}};
```



CLASSTIME Pg. No.
Date / /

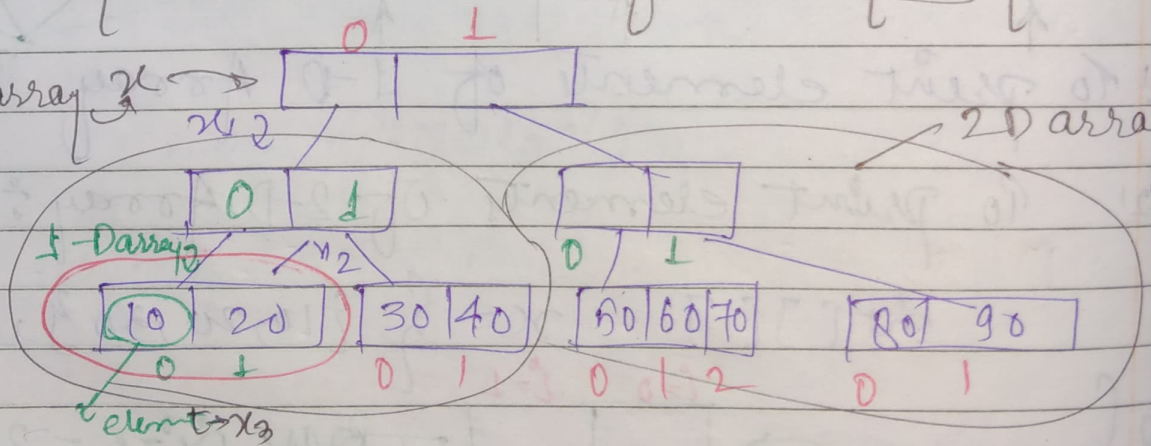
* Normal loop :-

```
for(int i=0; i<x.length; i++)
{
    for(int j=0; j<x[i].length; j++)
    {
        sopln(x[i][j]);
    }
}
```

* For each loop :-

```
for(int[] x1 : x)
{
    for(int x2 : x1)
    {
        sopln(x2)
    }
}
```

Ex:- To print elements of 3-D Array :-




```
for(int[][] x1 : x)
{
    for(int[] x2 : x1)
```

```
{
    for (int x2 : x2)
    {
        sopln (x2);
    }
}
```

Limitation of for-each loop :-

(1). For each loop is a best choice to retrive elements of arrays and collections. but its a limitation is it is applicable only for arrays & collections. and it is not a general purpose loop.

eg:-
```
for (int i=0; i<10; i++)
{
    sopln ("Hello");
}
```
⇒ We can't write an equivalent for-each loop directly.

(2) By using normal for loop we can print array elements either in original order or in reverse order.
But by using for each loop we can print array elements only in original order but not in reverse order.

eg:-
```
int [] x = {10, 20, 30, 40, 50};
for (int i = x.Length-1; i>=0; i++)
```

CLASSTIME | Pg. No.
Date / /

```
{
    for (x[i]);
}
```

⇒ We can't write an equivalent for-each loop directly.

Op:
50
40
30
20
10

Iterable (I) → Interface. — related to for each loop.

```
for (eachitem x : target)
{
    ≡
}
```

target → Iterable object
Array/collection

Iterable Interface :-

```
for (eachitem x : target)
{
    ≡
}
```

target → array/collection
↳ Iterable object.

The target element in for-each loop should be iterable object.

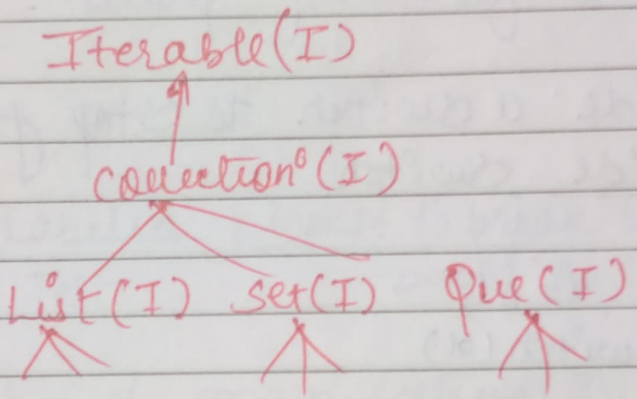
An object is said to be Iterable if corresponding class implements Java.lang.Iterable(I). interface

Iterable interface introduced in 1.5 V.

and it Contains Only One method. Iterator()

```
public Iterator iterator()
       returntype  method
```

All array related classes and Collection implemented classes already implemented iterable interface. Being a programmer we are not required to do anything but we should aware the part.



Differences B/w Iterator & Iterable :

	Iterator (I)	Iterable (I)
1.	It is related to collections	1. It is related to for-each loop.
2.	We can use to retrive the elements of Collection One By One.	2. The target element in for-each loop should be Iterable.
3.	Java.util pkg	3. Java.lang pkg
4.	3 methods 1. hasNext() 2. next() 3. remove()	4. 1 method iterator()

CLASSTIME Pg. No.
Date / /

Lecture – 29 → Break & Continue :-

Transfer statement :-
1- Break
2- Continue

1. Transfer statements is used to transfer the control from one place to another.

1- **Break :-** We can use break statement in the following places.

1- Inside a switch to stop fall through.

Inside switch

```
int x = 0;
switch (x)
{
    case 0:
        soph (0);
    case 1:
        soph (1);
        break;
    case 2:
        soph (2);
    default:
        soph ("def");
}
```

O/p:

```
0
1
```

CLASSTIME Pg. No.
Date / /

Lecture – 29 → Break & Continue :-

Transfer statement :-
1- Break
2- Continue

1. Transfer statements is used to transfer the control from one place to another.

1- **Break :-** We can use break statement in the following places.

1- Inside a switch to stop fall through.

Inside switch

```
int x = 0;
switch (x)
{
    case 0:
        sopln(0);
    case 1:
        sopln(1);
        break;
    case 2:
        sopln(2);
    default:
        sopln("def");
}
```

O/p:

```
0
1
```

2. Inside Loops: To break loop execution based on some condition.

```
for (int i=0; i<10; i++)
{
    if (i==5)
    break;
    sopln(i);
}
```

O/p:-
0
1
2
3
4.

3. Inside Labeled blocks: To break block execution based on some condition

```
class Test
{
    p s v main (String[] args)
    {
        int x=10;
        l1:
        {
            sopln ("begin");
            if (x==10)
            break l1;
            sopln ("end");
        }
        sopln ("Hello");
    }
}
```

O/p: Begin
Hello.

These above 3 are the only places where we can use break statement if you are using anywhere else we will get CTE. saying :— break outside switch or loop

eg :-

```
class Test
{
    p s v main(String[] args)
    {
        int x = 10;
        if (x==10)
            break;
        Sopln("hello");
    }
}
```

(2) Continue statement :- To skip current iteration and continue next iteration.

We can use continue statement inside loops to skip current iteration and continue for the next iteration.

eg :-

```
for (int i=0; i<10; i++)
{
    if (i%2==0)
        continue;
    Sopln(i);
}
```

CLASSTIME Pg. No.
Date / /

op:
1
3
5
7
9

l1:
{
=
=
}

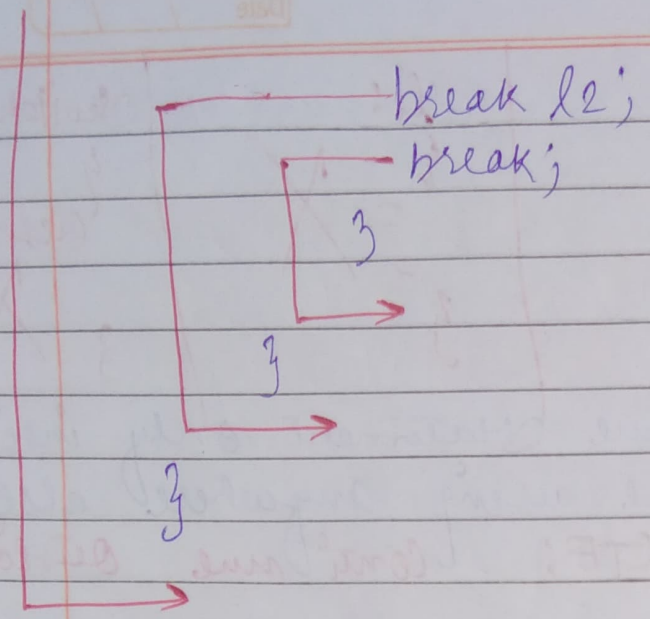
switch
{
case m
}

We can use continue statement only inside a loops if we are using anywhere else we will get CTE: continue outside of loop.

eg:
```
class Test
{
    p s v main(String[] args)
    {
        int x=10;
        if(x==10)
        continue;
        sopln("Hello");
    }
}
```

* Labelled break & continue :- We can use labelled break and continue to break or continue a particular loop in nested loops.

```
l1:
for(------>
{
    l2:
    for(------)
    {
        for(----)
        {
            break l1;
```



Que+

```
l1:
for (int i=0; i<3; i++)
{
    for (int j=0; j<3; j++)
    {
        if (i==j)
            continue l1;
        sopln (i+" -- "+j);
    }
}
```

Op

```
break;
1 --- 0
2 --- 0
2 --- 1
```

Op

```
break l1;
```

No O/P

Op

```
continue;
0 -- 1
0 -- 2
1 -- 0
1 --- 2
2 --- 0
2 --- 1
```

Op

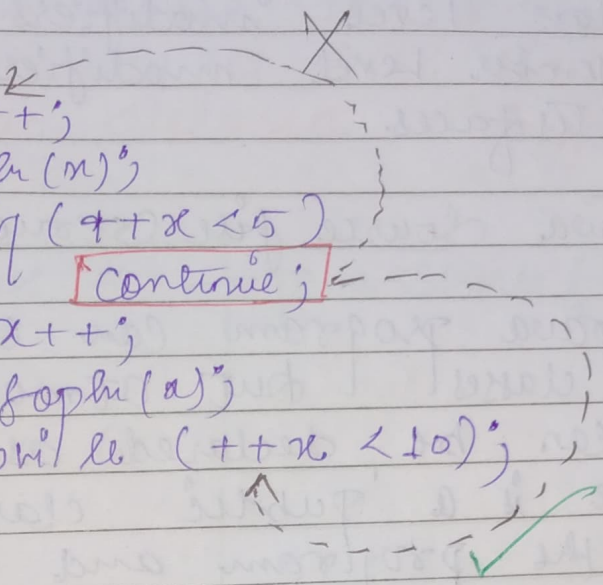
```
Continue l1;
1 -- 0
2 -- 0
2 -- 1
```

CLASSTIME Pg. No.
Date / /

do-while() Vs Continue [dangerous combination]

eg:

```
int x = 0;
do
{
    x++;
    printf(x);
    if (++x < 5)
        continue;
    x++;
    printf(a);
} while (++x < 10);
```



Op:

1
4
6
8
10

x = 0
1
2
3
4
5 6 7 8 9 10 11